

爱创课堂前端培训

JS 基础

第 1 天课堂笔记（本课程共 8 天）

班级：北京前端训练营 7 期

日期：2017 年 5 月 9 日

目录

JS 基础.....	1
目录.....	1
一、 javascript 简介.....	2
1.1 用途：	2
1.2 Javascript 简史.....	3
1.3 ECMAScript 版本发展.....	4
1.4 ECMAScript 脚本语言.....	4
二、 hello world.....	4
2.1 js 书写位置.....	4
2.2 js 注释.....	5
2.3 alert 语句.....	5
2.4 控制台.....	6

三、 字面量.....	7
3.1 数字字面量.....	7
3.2 字符串字面量.....	10
四、 变量.....	10
4.1 体会变量.....	10
4.2 变量声明.....	11
4.3 变量赋值.....	12
4.4 变量声明提升.....	12
4.5 同时定义多个变量.....	13
五、 数据类型.....	13
5.1 数据类型的检测.....	13
5.2 数据类型的转换.....	14
5.2.1 数字类型转换为字符串类型.....	14
5.2.2 字符串转换为数字.....	14

一、javascript 简介

前端分为 3 层：

结构层	作用：从 html 角度搭建网页
样式层	作用：从美化角度书写样式
行为层	作用：从交互的角度描述我们的页面行为

1.1 用途：

数据验证

读写 HTML 元素

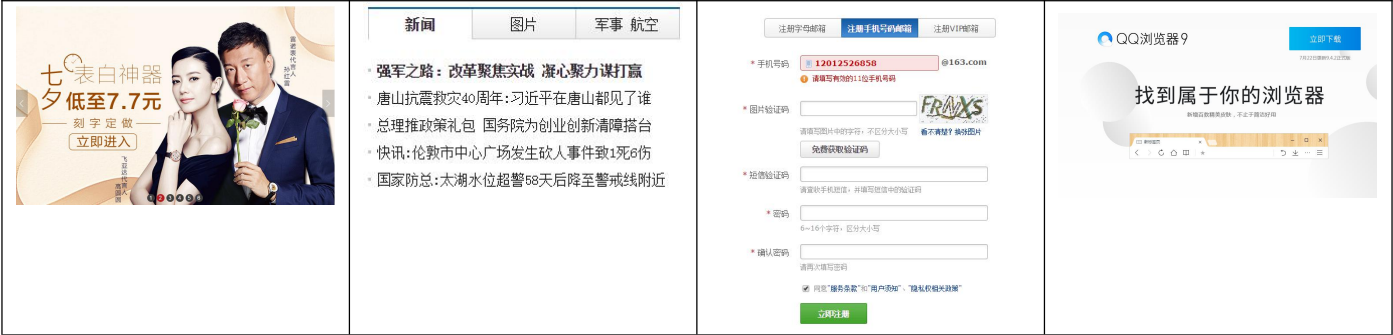
与浏览器窗口及其内容的交互效果

网页特效

WEB 游戏制作

基于 Node.js 技术进行服务器端编程

举例：



Javascript 用途很多，我们随时学习随时积累。

了解 Javascript 发展史。

1.2 Javascript 简史

在 WEB 日益发展的同时，网页的大小和复杂性不断增加，受制于网速的限制，为完成简单的表单验证而频繁地与服务器交换数据只会加重用户的负担，当时走在技术革新最前沿的 Netscape（网景）公司，决定着手开发一种客户端语言，用来处理这种简单的验证。

1995 年，就职于 Netscape 公司的布兰登·艾奇（Brendan Eich），开始着手为即将于 1996 年 2 月发布的 Netscape Navigator 2 浏览器开发一种名为 LiveScript 的脚本语言。为了尽快完成 LiveScript 的开发，Netscape 与 Sun 公司建立了一个开发联盟。在 Netscape Navigator 2 正式发布前夕，Netscape 为了搭上媒体热炒 Java 的顺风车，临时把 LiveScript 改名为 JavaScript。

由于 JavaScript1.0 获得的关注度越来越高，1996 年，微软就在其 Internet Explorer 3 中加入了名为 JScript 的 JavaScript 实现，这意味着有了两个不同的 JavaScript 版本，导致 JavaScript 没有一个标准化的语法和特性。1997 年，以 JavaScript 1.1 为蓝本的建议被提交给了欧洲计算机制造商协会（ECMA，European Computer Manufacturers Association）。该协会指定 39 号技术委员会（TC39，Technical Committee #39）负责“标准化一种通用、跨平台、供应商中立的脚本语言的语法和语义”。TC39 由来自 Netscape、Sun、微软、Borland 及其他关注脚本语言发展的公司的程序员组成，他们经过数月的努力完成了 ECMA-262 标准，定义一种名为 ECMAScript 的新脚本语言。

布兰登·艾奇（1961 年～），JavaScript 的发明人，目前（2005 年至 2014 年）在 Mozilla 公司担任 CTO。2014 年 4 月 3 日，出任 Mozilla 的 CEO 十天就被迫辞职。



1.3 ECMAScript 版本发展

1998 年 6 月, ECMAScript 2.0 版发布。

1999 年 12 月, ECMAScript 3.0 版发布, 成为 JavaScript 的通行标准, 得到了广泛支持。

2007 年 10 月, ECMAScript 4.0 版草案发布, 对 3.0 版做了大幅升级。草案发布后, 由于 4.0 版的目标过于激进, 各方对于是否通过这个标准, 发生了严重分歧。以 Yahoo、Microsoft、Google 为首的大公司, 反对 JavaScript 的大幅升级, 主张小幅改动; 以 JavaScript 创造者 Brendan Eich 为首的 Mozilla 公司, 则坚持当前的草案。

2008 年 7 月, 由于各方分歧太大, 争论过于激进, ECMA 开会决定, 中止 ECMAScript 4.0 的开发, 将其中涉及现有功能改善的一小部分, 发布为 ECMAScript 3.1, 而将其他激进的设想扩大范围, 放入以后的版本, 由于会议的气氛, 该版本的项目代号起名为 Harmony (和谐)。会后不久, ECMAScript 3.1 就改名为 ECMAScript 5。

2009 年 12 月, ECMAScript 5.0 版正式发布。Harmony 项目则一分为二, 一些较为可行的设想定名为 JavaScript.next 继续开发, 后来演变成 ECMAScript 6; 一些不是很成熟的设想, 则被视为 JavaScript.next.next, 在更远的将来再考虑推出。

2011 年 6 月, ECMAScript 5.1 版发布, 并且成为 ISO 国际标准 (ISO/IEC 16262:2011)。

2013 年 3 月, ECMAScript 6 草案冻结, 不再添加新功能。新的功能设想将被放到 ECMAScript 7。

2013 年 12 月, ECMAScript 6 草案发布。然后是 12 个月的讨论期, 听取各方反馈。

2015 年 6 月 17 日, ECMAScript 6 发布正式版本, 即 ECMAScript 2015。

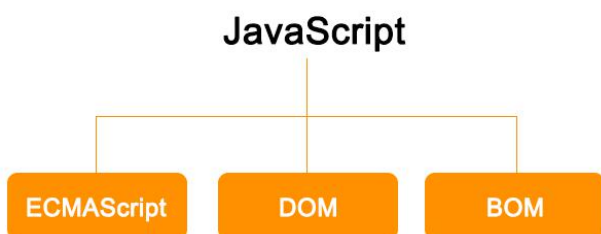
2015 版本也就是 6 版本, 将来在我们最后面的课程。

1.4 ECMAScript 脚本语言

JavaScript, JScript, ActionScript 等脚本语言都是基于 ECMAScript 标准实现的。

在 JavaScript, JScript 和 ActionScript 中声明变量, 操作数组等语法完全一样, 因为它们都是 ECMAScript。但是在操作浏览器对象等方面又有各自独特的方法, 这些都是各自语言的扩展。

JavaScript 是由 ECMAScript, DOM 和 BOM 三者组成的。



二、hello world

2.1 js 书写位置

①可以直接书写在 body 内部一对 script 标签里, script 标签有一个 type 属性, 属性值“text/javascript”表示标签内部书写是纯文本的 javascript 语言。

javascript 通常都简称为 js.

js 必须依托 html 载体在浏览器中呈现, 不能脱离 html 单独在浏览器中呈现。

```
1 <script type="text/javascript">
2     javascript 代码
3 </script>
```

②写在 head 标签内部。这是实际工作中使用的方法。

```
1 <head>
2     <meta charset="UTF-8">
3     <title>Document</title>
4     <script type="text/javascript" src="02_js.js"></script>
5 </head>
```

2.2 js 注释

```
1 <!-- html 注释 -->
2 /*css 注释*/
```

单行注释: (快捷键: **ctrl+/****)

```
1 //alert("hello world");
2 //我是单行注释
3 后面再写就不行了
```

多行注释: (快捷键: **ctrl+shift+/****)

```
1 /*
2 我是多行注释
3 我可以书写多行
4 我不在浏览器中渲染
5 */
```

2.3 alert 语句

alert 本身表示“示警, 提示” alert 在 js 中表示提示框, 弹出框。alert 语句是 js 内置功能。

①alert 书写

alert 后面紧跟小括号, 小括号里面必须书写**一对双引号**。

```
1 alert('这是错误写法')
```

②alert 语句中分号的重要性

要求大家必须每一句 **alert 语句后面必须写分号**。

```
1 alert("这是第一句话");
2 alert("2 句话")alert("3 句话")alert("4 句话")
```

js 解析器的原理: js 解析器在解析时, 一句话一句话进行解析, 当看到**分号时**, 知道这是一句 js 的结束。
没有分号解析器会一直解析, 直到遇见换行, 也会认为这是 js 语句结束。

但是多个 alert 不写分号, 解析器不能正常解析。

下面这段代码可以正常解析: (因为每一句 alert 语句有换行。)

```
1 alert("这是第一句话");
2 alert("2 句话")
```

```
3 alert("3 句话")
4 alert("4 句话")
```

虽然上面代码可以正常解析，但是因为 js 在提交之前，会进行压缩。会去掉所有换行，空格等。变成下面的效果，解析器不能正常解析。

```
1 alert("这是第一句话");alert("2 句话")alert("3 句话")alert("4 句话")
```

作用：为了让解析器正常工作，**同时提高解析器的效率**。

③alert 语句的解析顺序

通常情况下，是从上到下依次解析。

```
1 alert("one");
2 alert("two");
3 alert("three");
4 alert("four");
5 alert("one");
6 alert("two");
7 alert("three");
8 alert("four");
9 alert("one");
10 alert("two");
11 alert("three");
12 alert("four");
```

④alert 语句对空格、缩进、换行不敏感。

```
1 alert("one");alert("two");      alert("three");
2 alert("four");
3 alert("one");      alert("two");
```

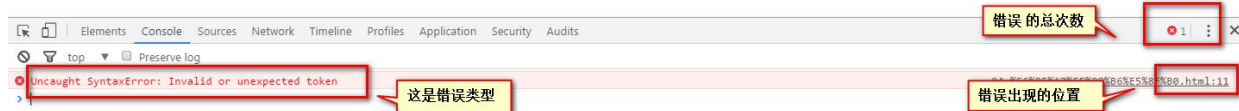
2.4 控制台

通常我们使用谷歌的控制台，通过“审查元素”或者是“检查”可以调出控制面板，控制面板里有一个 console（控制台），或者是快捷键 F12。

作用：①汇总我们 js 出现错误代码。

指明错误代码的类型，错误出现的位置，错误的总次数。

```
1 alert("我是第一条语句"); //因为使用了中文符号
```



方便程序员进行代码调试。

②可以在控制面板打印语句。

console 也是 js 内置的对象，console 有一个方法 log（日志）使用 console.log() 进行语句输出。

可以直接在控制台进行输出。

直接写 **console.log** 后面紧跟小括号，小括号里书写**一对双引号**，双引号里书写你想要输出的内容。按回车键进行输出。

```

> console.log("第一句话")
第一句话
undefined
> console.log("第二句话")
第二句话
undefined
>

```

还可以在代码中书写。直接在控制台输出语句

```

1 console.log("这是 1");
2 console.log("这是 2");
3 console.log("这是 3");
4 console.log("这是 4");
5 console.log("这是 5");
6 console.log("这是 6");
7 console.log("这是 7");
8 console.log("这是 8");

```

在控制台输出的效果:

```

这是1
这是2
这是3
这是4
这是5
这是6
这是7
这是8
>

```

三、字面量

字面量：是表示**固定值**的一种表示方法。字面量的字面含义就是你看到什么就是什么。

字面量也叫常量。

字面量：数字，字符串，`undefined`，布尔类型的值。今天只学数字字面量和字符串字面量。

3.1 数字字面量

数字字面量包含：整数，浮点数（小数），特殊值。

①整数：

十进制表示法，八进制表示法，十六进制表示法。

八进制表示法：前缀 `0o,0O`，后面数字 `0-7`。

十六进行表示法：前缀 `0x,0X`，后面数字 `0-9`，字母 `a-f` 或者 `A-F`

在进行算术运算时，都会转换成十进制。（控制台输出的是十进制的数）

八进制：**逢八进一**。

```

1 console.log(010);
2 console.log(0100);
3 console.log(01000);
4 console.log(010000);

```

```

8
64
512
4096
>

```

八进制每一位数都不能超过 7，如果超过了会强制转换成十进制。

```
1 console.log(087);
```

前缀是 0o 或者 0O 会直接报错。

```
1 console.log(0o87);
2 console.log(0O87);
```



十六进制：逢十六进一

```
1 console.log(0X10);
2 console.log(0x100);
3 console.log(0x1000);
4 console.log(0xff);
```

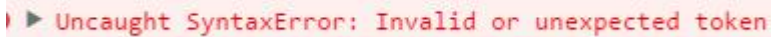
```

16
256
4096
255
>

```

十六进制每一位数不能超过 0-9 或者 a-f (A-F)，超过直接报错。

```
1 console.log(0xgh);
```



以下这些都输出 15

```
1 console.log(15);
2 console.log(017);
3 console.log(0o17);
4 console.log(0017);
5 console.log(0Xf);
6 console.log(0xf);
```

```

15
15
15
15
15
15

```

以下这些都输出-15

```
1 console.log(-15);
2 console.log(-017);
```



```

3 console.log(-0o17);
4 console.log(-0017);
5 console.log(-0Xf);
6 console.log(-0xf);

```

```
-15
```

```
-15
```

```
-15
```

```
-15
```

```
-15
```

```
-15
```

②浮点数字面量

整数.小数（浮点数只能用十进制表示）

表示：

```

1 console.log(22.33);
2 console.log(-1.8);
3 console.log(0.893);
4 console.log(.893);
5 console.log(1.67809e3);
6 console.log(0.567e-3);

```

```
22.33
```

```
-1.8
```

```
0.893
```

```
0.893
```

```
1678.09
```

```
0.000567
```

js 也认识幂的写法。着重记性。

③特殊值

Infinity，表示无穷

浏览器对于数值计算能力有限，有一个最大值超过最大值显示(Infinity)，有一个最小值，超过最小值显示(-Infinity)

```

1 console.log(892e1238949859589686986);
2 console.log(-892e1238949859589686986);

```

```
Infinity
```

```
-Infinity
```

这种也是无穷

```

1 console.log(1/0);
2 console.log(-1/0);

```

```
Infinity
```

```
-Infinity
```

NaN，表示 not a number 很奇怪，但是却表示数字字面量。

```

1 console.log(NaN);
2 console.log(0/0);

```

```
NaN
```

```
NaN
```

3.2 字符串字面量

字符串字面量，就是平时生活中说的话。包括各种类型的语言，特殊符号，说话中的数字。

表示方法：必须用一对引号包裹（双引号“”，单引号”）。必须是同类的引号。（数字字面量不用任何符号）

```
1 console.log("你今年多大了? ");
2 console.log("18");
```

字符串字面量可以有零个字符或者是多个字符。

特殊字符：

\n 表示换行

\t tab 表示制表符

```
1 console.log("今天\n 是周二， \n5 月 9 日");
```

```
今天
是周二，
5月9日
```

特殊符号书写：使用反斜杠+符号

\" 表示双引号

' 表示单引号

\\

```
1 console.log("我想输出一些符号比如双引号\",或者是反斜杠\\。");
```

四、变量

变量（variables），可以把变量看成一个容器，里面可以存放任何数据，包括字面量，函数，数组等等各种数据。

4.1 体会变量

```
1 // 定义变量
2 var a;
3 // 变量赋值
4 a = 10;
5 // 引用变量
6 console.log(a);
```

我们变量输出时，输出的是变量存放的数据，而不是变量名。

变量只能存放一个数据，当有多个赋值时，会舍弃掉旧的。保存最后一个赋值。

```
1 var a;
2 // 变量赋值
3 a = 10;
4 a = "hello world";
5 // 引用变量
6 console.log(a);
7
```

```
hello world
```

4.2 变量声明

变量声明时，必须使用关键字 **var**，var 后面紧跟一个空格，空格后面在写变量名。

```
1 var b;
```

变量名也是标识符。标识符有自己的命名规则。

第一个字符可以是字母，下划线（ ）或者美元符号（\$）

其他字符可以是字母，下划线，美元符号，或者数字。

合法的标识符：

```
1 var a_;
2 var _____;
3 var $_$;
4 var $_1111;
```

不合法的标识符：

```
1 var 1_a;           //不能数字开头
2 var $_&*;          //不能有&*这些特殊字符
3 var a b;           //字符之间不能有空格
```

js 严格区分大小写字母，所以我们上面说的字母包括 a-z 也包括 A-Z。

标识符也不能使用 js 规定的关键字，保留字。

关键字：就是有特殊功能的单词。

保留字：就是 js 保留下来，将来会成为关键字。

关键字：

```
break    do    instanceof    typeof    case    else    new    var    catch    finally
return    void    continue    for    switch    while    debugger*    function    this
with      default    if    throw    delete    in    try
```

保留字：

```
abstract    enum    int    short    boolean    export    interface    static    byte
extends    long    super    char    final    native    class
synchronized    float    package    throws    const    goto    private    transient
debugger    implements    protected    volatile    double
import    public
```

变量如果不定义直接使用，会抛出一个“引用错误”。

```
1 var a;
2 a = 10;
3 console.log(a);
4 console.log(c);
```



引用错误

4.3 变量赋值

变量在声明之后直接引用会输出“undefined”表示未定义。表示该变量可以存放任何类型的数据。

```
1 var a;
2 console.log(a);
```

```
undefined
```

js 中用=等号表示赋值。表示把右边的数据赋值给左边的变量。

```
1 var a;
2 var b;
3 a = 3;
4 b = a;
5 a = 4;
6 console.log(a);
7 console.log(b);
```

```
4
```

```
3
```

变量赋值时，左边变量值改变，右边的值不变。

```
1 var a;
2 var b;
3 a = 3;
4 b = a;
5 a = b + 3;
6 console.log(a);
7 console.log(b);
```

通常我们会将变量的声明和赋值写一起：

```
1 var a = 10;
```

4.4 变量声明提升

js 有一个非常好的特性，当引用后面声明的一个变量时，不会抛出错误。只是输出 **undefined**。这就是变量声明的提升。

```
1 console.log(a);
2 var a = 10;
```

```
undefined
```

js 中，当引用一个前面未定义的变量，但是后面进行定义。这时会输出 **undefined**。即使你后面给变量进行

了赋值，仍然是输出 **undefined**。就好像是 js 中所有的变量的声明被“举起”，提升到所有语句之前。

原理：js 在解析时，会先查看整个 js 语句，将所有的变量声明提升到 **js 语句的最前面**。（预解析）

变量只提升变量的声明，不提升变量的赋值。

等同于：

```
1 var a;
2 console.log(a);
3 a = 10;
```

4.5 同时定义多个变量

一个 var 可以同时定义多个变量，变量之间用**逗号**隔开。

```
1 var a,b,c;
2 a = 1;
3 b = 2;
4 c = 3;
5 console.log(a);
6 console.log(b);
7 console.log(c);
```

或者是定义和赋值写一起。

```
1 var a = 1, b = 2, c = 3;
2 console.log(a);
3 console.log(b);
4 console.log(c);
```

五、数据类型

不管我们学习字面量，变量，以及将要学数组，函数等，都有自己的类型。

简单数据类型：5 类

number：数字类型，就是我们学的整数，浮点数，特殊值（Infinity 和 NaN）。

string：字符串类型。就是字符串。

undefined:undefined 类型，这个类型就一个值，就是自己。（表示未定义时值。）

Boolean: boolean 类型。就两个值 true 和 false。

null：后面会说明。

复杂的类型：

object ：对象

5.1 数据类型的检测

typeof ：操作符，typeof()后面的括号可有可无。

```
1 console.log(typeof 10);
2 console.log(typeof "10");
3 console.log(typeof 2.7878);
4 console.log(typeof Infinity);
5 console.log(typeof "你好");
6 console.log(typeof true);
7 console.log(typeof NaN);
```

```

number
string
number
number
string
boolean
number

```

变量的数据类型: js 是动态语言, 变量的数据类型也是变化的。取决于变量的赋值。赋值是什么类型变量就是什么类型。

```

1 var a = 10;
2 var b = "hello world";
3 console.log(typeof a);
4 console.log(typeof b);

```

```

number
string

```

5.2 数据类型的转换

5.2.1 数字类型转换为字符串类型

加号: 在 js 中尤其是不同类型数据之间的加号表示数据连接 (**拼接**)。特别注意是数字和字符串之间的加号就是拼接。

只有当加号两边都是数字时才表示数学运算的加法。

```

1 var a = 1;
2 var b = "1";
3 console.log(a + b);
4 console.log(typeof(a + b));

```

```

11
string

```

数字转换为字符串的方法: 数字 + "" ; (数字加空字符串)

```

1 var c = 12;
2 c = c + "";
3 console.log(typeof c);

```

```

string

```

5.2.2 字符串转换为数字

prompt 类似于 alert 也是弹出框。它可以进行文字输入。

prompt 有两个参数, 第一个参数表示提示文本。第二个参数表示默认文本 (可以省略)。两个参数直接用逗号隔开。

```

1 prompt("请输入您的年龄", 18);

```

prompt 输入的数据可以用变量接收。

prompt 输入的数据类型都是 **string**。

```

1 var a = prompt("请输入您的年龄");
2 console.log(a);

```

```
3 console.log(typeof a);
```

```
99
string
```

小案例:

```
1 var a = prompt("请输入第一个数字");
2 var b = prompt("请输入第二个数字");
3 console.log(a + b);
4 console.log(typeof(a + b));
```

字符串转换为数字有 3 中方法 `Number()`, `parseInt()`, `parseFloat()`。 `Number()` 方法可以进行任何数据的转化。 `parseInt()` 和 `parseFloat()` 这两种方法只能进行字符串的转换。

`parseInt()` 将字符串转化为数字类型。

```
1 console.log(parseInt("12.999999"));
2 console.log(parseInt("12 今天天气不错"));
3 console.log(parseInt("今天天气不错 12"));
4 console.log(parseInt("0.11111 你好"));
```

```
12
12
NaN
0
```

进制的问题:

另外, 由于 ECMAScript 3 和 5 存在着分歧, 调用 `parseInt()` 函数时最好总是带上进制(radix) 参数, 这个参数用于指定使用哪一种数制。如果指定了进制, 那字符串前可以不带前缀“0”、“0o”、“0x”。

`parseInt` 也有两个参数, 第一参数是要转的字符串, 第二个是进制。

```
1 console.log(parseInt("10",8));
2 console.log(parseInt("10",16));
3 console.log(parseInt("10",10));
```

以下代码都输出 15;

```
1 console.log(parseInt("15",10));
2 console.log(parseInt("f",16));
3 console.log(parseInt("0xf",16));
4 console.log(parseInt("0Xf",16));
5 console.log(parseInt("17",8));
6 console.log(parseInt("12",13));
7 console.log(parseInt("15 基金勳加高万几个几个万国际公法加工费",10));
```

```
15
15
15
15
15
15
15
```

parseInt 作用：可以对数字进行**取整**。

parseFloat():将字符串转换为浮点数。

```
1 var a = prompt("请输入一个浮点数");
2     var b = parseFloat(a);
3     console.log(a);
4     console.log(b);
5     console.log(typeof b);
```

12.111

12.111

number

```
1 console.log(parseFloat("12.1111111"));
2     console.log(parseFloat("0.278895"));
3     console.log(parseFloat(".278895"));
4     console.log(parseFloat(".278895e3"));
5     console.log(parseFloat(".12 你好 0.123"));
6     console.log(parseFloat("你好 0.123"));
```

12.1111111

0.278895

0.278895

278.895

0.12

NaN

parseFloat 直接忽略第一个浮点数后面的所有东西。

```
1 console.log(parseFloat("0.1234 你好世界"));
2 console.log(parseFloat("0.1234.1234"));
3 console.log(parseFloat("0.1234e2.234"));
```

0.1234

0.1234

12.34

1